Log Monitoring Workflow

Author: Sukhvir Kaur Sahota

**Executive Summary**

In the Cybersecurity industry, Log monitoring plays a crucial role. By monitoring and logging the traffic , businesses can reduce its cyber attack risk. A good cyber monitoring system will give you real-time insight into your network traffic, alert you to potential threats, and respond to intrusions quickly.(1)

This report shows how the New Leaf organization is working on avoiding any vulnerabilities which could lead to cyber attack in their organization. The report is focused on scripts built to analyze logs, set up thresholds and send email alteration and reports. We used a Python script to perform the log monitoring and used some manual commands as tools such as grep. awk, sort, cat, sudo. The goal here is to find out the number of failed attempts and potential risk associated with it. By finding out unusual behavior the New leaf can work on improving their network security to avoid/reduce attacks such as Brute force, DDoS and Data manipulation.

Although the approach used here seems nice it might not be sufficient with business growth, for that we have suggested they use Visualization tools, Do real time monitoring and get trained on current affairs whenever possible.

**Introduction**

In this report we will discuss Log Monitoring Workflow required by Turn a New Leaf company which is a non profit organization that supports youth in a range of rural communities to seek employment. The workflow will show key things such as how to monitor unusual traffic, commands and scripts to use, expected output as well as future scope of skill development to make the workflow more efficient.

Our Organisation New Leaf uses Linux and Windows machines and 2 web servers in their network. In order to support government regulatory requirements, their members must log into the company system every Thursday to confirm or update their employment status, and input any updates on their job searches, including links to job listings they are actively pursuing.(2)

Our Objective here is to Monitor access logs for unusual network traffic and failed login attempts, focusing on Thursdays due to regulatory requirements. Send weekly reports and alerts to the manager for potential security concerns.

**Workflow**

For this Scenario we can have the following workflow structure.(3)(4)



1. ***Data Collection*** : Monitor access logs daily from Both machines and/or web server.
2. ***Data Processing*** : Implementation of script (such as Python or bash Script)

* Parse logs for Identification of failed login attempts
* Set Threshold on unusual no. of failed login attempts

1. ***Alert Generation*** : Send an email to the manager if the number of failed attempts exceeds the predefined threshold.
2. ***Weekly Report Update***: Compile a report which shows login attempts, unusual behavior and maybe daily traffic.
3. ***Documentation*** : Maintain a document which shows

* Workflow steps and explanation
* Script and commands
* Unusual behavior

Update the document when new threats are identified or if there is any changes in process

**Programming**

For this Report, I am using following

**Log Analysis Tools:** grep, awk, and sed for parsing and extracting relevant information from the access logs.

**Key Scripts:**

* 1. Data parsing script to extract relevant information from log files.
  2. Script to calculate baseline traffic patterns and identify anomalies.
  3. Script to generate automated email reports and send alerts.

**Steps/commands to create Script in Linux**

1. Command to create a file in linux terminal

**$sudo nano <file\_name>.py**

1. Write a Python script

**#!/usr/bin/python3**

* The Python script should be start with line **#!/usr/bin/python3** is called a shebang line or hashbang. #!/usr/bin/python3 indicates that the script should be executed using the Python 3 interpreter located at /usr/bin/python3.
* Similarly, if you want to use bash script then #!/bin/bash

1. Save Script

To save script use **Ctrl +X**, Select ***Y*** to save and enter.

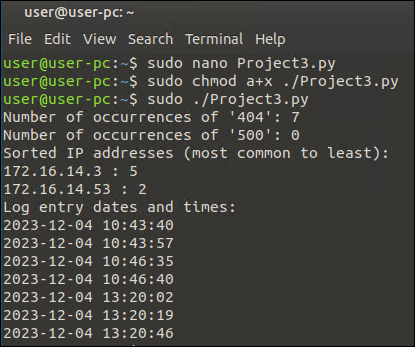
1. Command to give permissions to script

**$sudo chmod a+x ./<file\_name>.py**

1. To test if the script is fine we can test it

**$sudo ./<file\_name>.py**

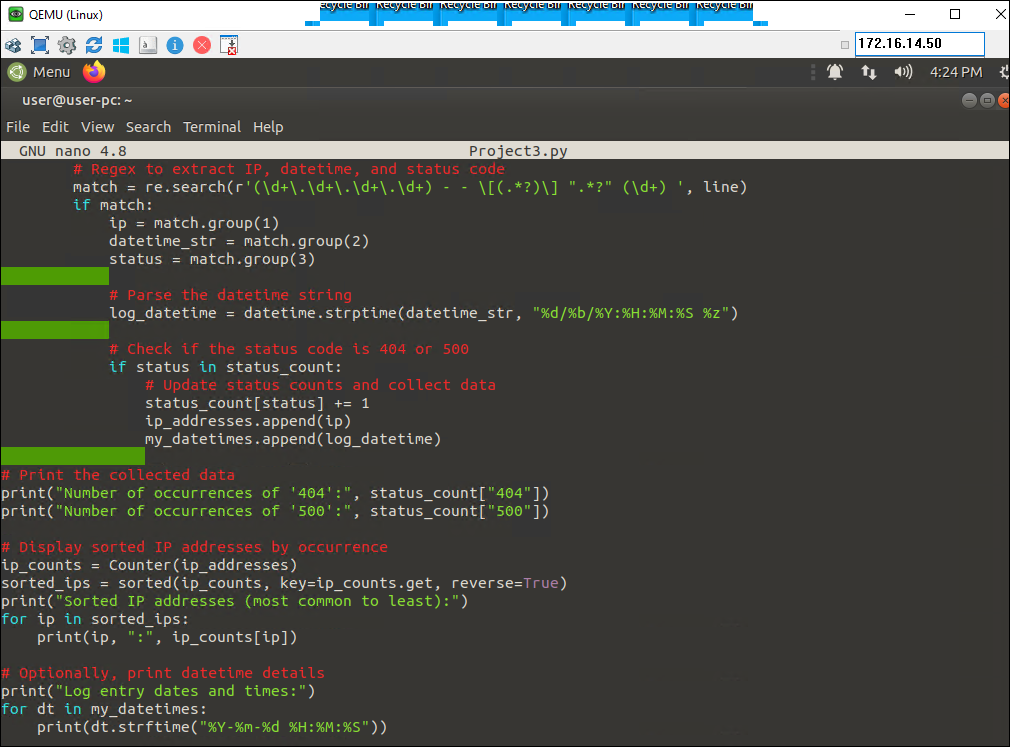
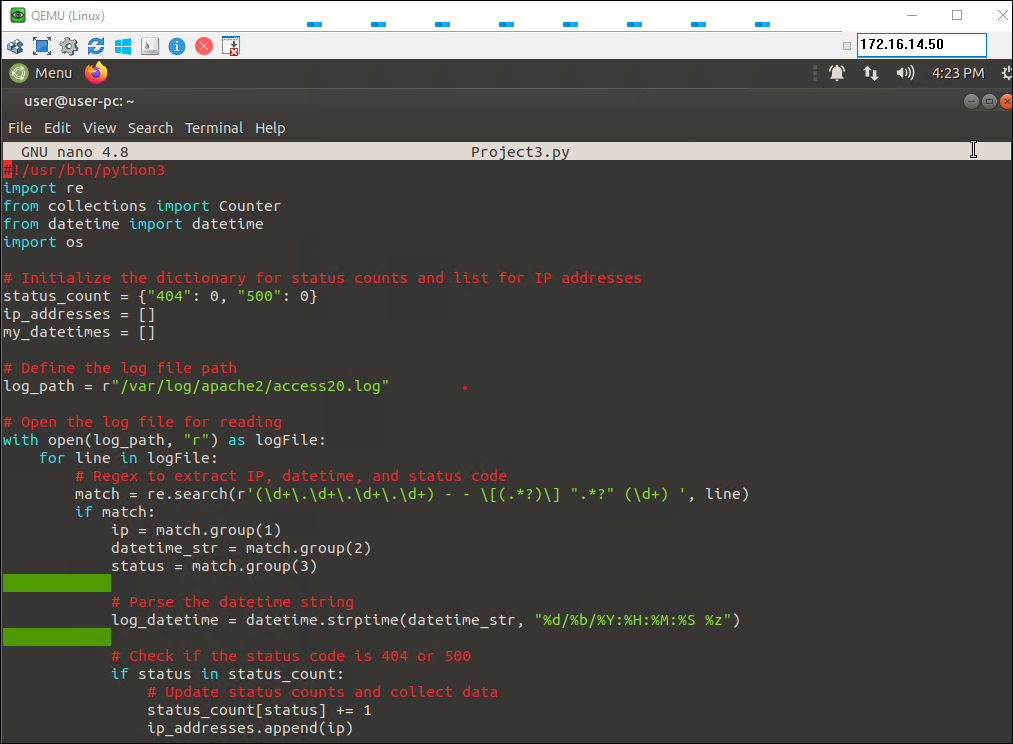
1. The following output is from access.logs which we retrieve from jumphost



**Script-1 :Data parsing script to extract relevant information from log files.**

This Script is doing the following (5)

1. Reading the log file form path /var/log/apache2
2. Doing RegEx to extract IP, datetime and status code
3. Parsing datetime string
4. Checking for status code 404 and 500
5. Print data



The purpose of this script is to analyze the apache access logs

1. Imports:

| import re  from collections import Counter  from datetime import datetime  import os |
| --- |

* 1. ***re***: It supports the regular expressions, which are used for pattern matching.
  2. ***Counter from collections***: It's a container that keeps track of how many times equivalent values are added.
  3. ***datetime from datetime***: Use to supply classes for manipulating dates and times.
  4. ***os***: It provides a portable way of using operating system-dependent functionality.(4)

1. Initialization:

| # Initialize the dictionary for status counts and list for IP addresses  status\_count = {"404": 0, "500": 0}  ip\_addresses = []  my\_datetimes = [] |
| --- |

* 1. ***status\_count***: A dictionary to store the counts of different HTTP status codes. Initialized with counts of "404" and "500" set to 0.
  2. ***ip\_addresses***: A list to store IP addresses extracted from the log file.
  3. ***my\_datetimes***: A list to store timestamps extracted from the log file.(4)

1. Log file path:

| # Define the log file path  log\_path = r"/var/log/apache2/access20.log" |
| --- |

* 1. log\_path: The path to the log file, located at /var/log/apache2/access20.log

1. Reading the log file:

| # Open the log file for reading  with open(log\_path, "r") as logFile:  for line in logFile: |
| --- |

* 1. The script opens the log file in read mode using a context manager (with open(log\_path, "r") as logFile:), ensuring that the file is properly closed after use.
  2. It then iterates through each line of the log file.(4)

1. Regex Matching:

| # Open the log file for reading  with open(log\_path, "r") as logFile:  for line in logFile:  # Regex to extract IP, datetime, and status code  match = re.search(r'(\d+\.\d+\.\d+\.\d+) - - \[(.\*?)\] ".\*?" (\d+) ', line)  if match:  ip = match.group(1)  datetime\_str = match.group(2)  status = match.group(3) |
| --- |

* 1. The script uses a regular expression (re.search) to match patterns in each line of the log file. The pattern matches an IP address, timestamp, and HTTP status code.
  2. If a match is found (if match:), it extracts the IP address, timestamp, and status code.(4)

1. Data Processing:

| # Parse the datetime string  log\_datetime = datetime.strptime(datetime\_str, "%d/%b/%Y:%H:%M:%S %z")    # Check if the status code is 404 or 500  if status in status\_count:  # Update status counts and collect data  status\_count[status] += 1  ip\_addresses.append(ip)  my\_datetimes.append(log\_datetime) |
| --- |

* 1. It parses the extracted timestamp into a datetime object using datetime.strptime() with the appropriate format.
  2. If the status code is either "404" or "500", it updates the corresponding count in the status\_count dictionary, and it adds the IP address and timestamp to their respective lists.(4)

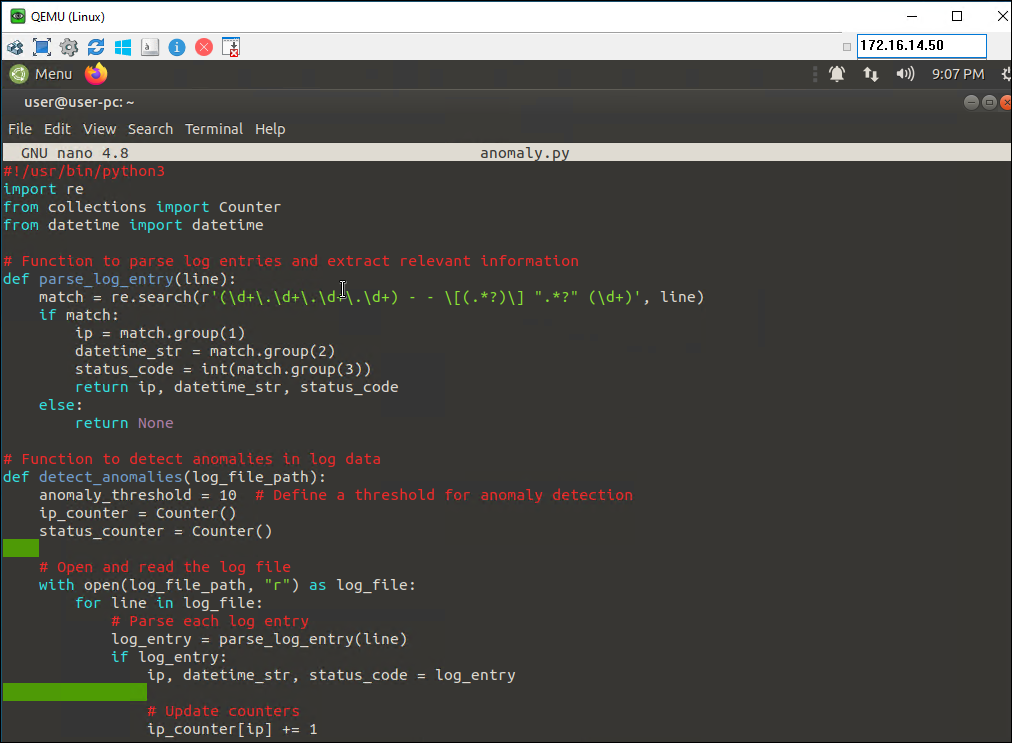
1. Output:

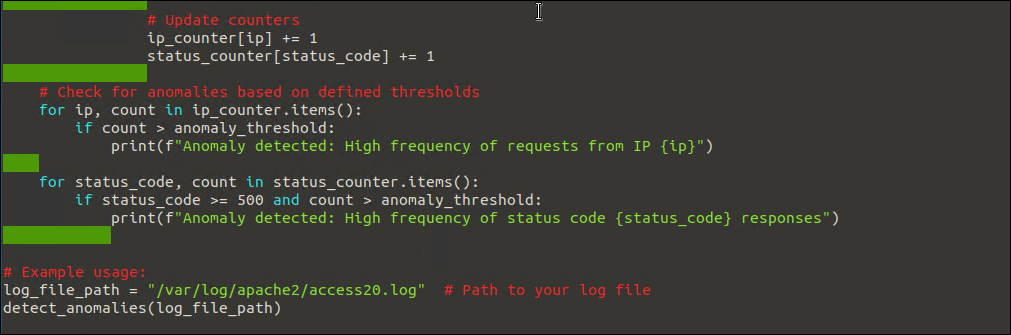
| # Print the collected data  print("Number of occurrences of '404':", status\_count["404"])  print("Number of occurrences of '500':", status\_count["500"])  # Display sorted IP addresses by occurrence  ip\_counts = Counter(ip\_addresses)  sorted\_ips = sorted(ip\_counts, key=ip\_counts.get, reverse=True)  print("Sorted IP addresses (most common to least):")  for ip in sorted\_ips:  print(ip, ":", ip\_counts[ip])  # Optionally, print datetime details  print("Log entry dates and times:")  for dt in my\_datetimes:  print(dt.strftime("%Y-%m-%d %H:%M:%S")) |
| --- |

* 1. Prints the counts of occurrences for HTTP status codes "404" and "500".
  2. Sorts the list of IP addresses by their occurrences in descending order and prints them.
  3. Optionally, it prints the extracted timestamps.(4)(5)

**Script 2 :To Identify anomalies(6)**

In addition to Script one, we can add a function to detect anomalies.





1. Function ():

| #Function to detect anomalies in log data  def detect\_anomalies(log\_file\_path):  anomaly\_threshold = 10 # Define a threshold for anomaly detection  ip\_counter = Counter()  status\_counter = Counter() |
| --- |

* detect\_anomalies(log\_file\_path): This is a function definition named detect\_anomalies which takes a single argument log\_file\_path, representing the path to the log file.
* anomaly\_threshold = 10: This variable sets the threshold for anomaly detection. An anomaly will be detected if the frequency of occurrence exceeds this threshold.
* ip\_counter = Counter(): This creates an empty counter object named ip\_counter. It will be used to count the occurrences of each IP address in the log data.
* status\_counter = Counter(): Similar to ip\_counter, this creates an empty counter object named status\_counter. It will count the occurrences of each HTTP status code in the log data.(6)(7)

1. Output form this function

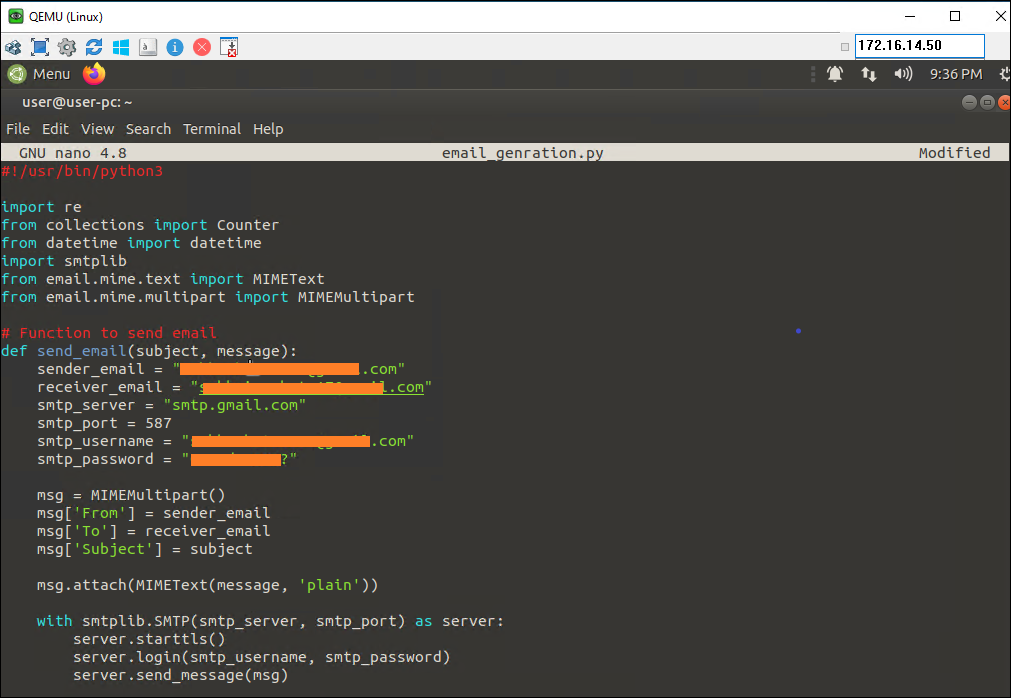
| #Check for anomalies based on defined thresholds  for ip, count in ip\_counter.items():  if count > anomaly\_threshold:  print(f"Anomaly detected: High frequency of requests from IP {ip}")  for status\_code, count in status\_counter.items():  if status\_code >= 500 and count > anomaly\_threshold:  print(f"Anomaly detected: High frequency of status code {status\_cod>  sudo awk '{print $1}' /var/log/apache2/access20.log | sort | uniq -c | awk '$1 > 10' |
| --- |

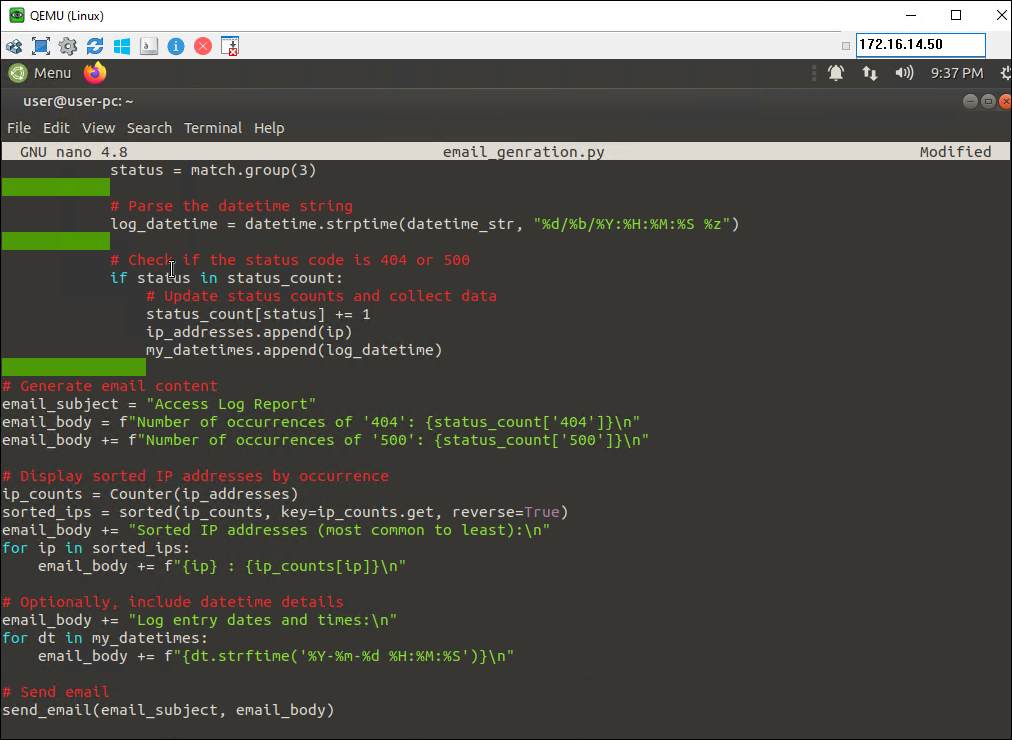
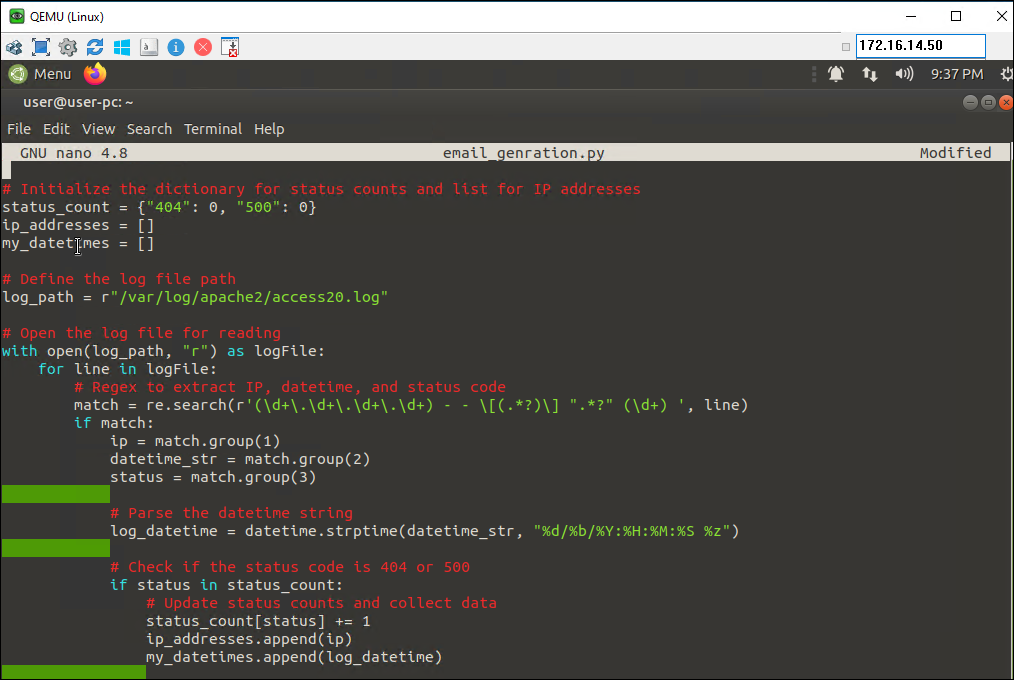
The for loops iterate over the items (IP addresses and status codes) in ip\_counter and status\_counter, but they don't perform any useful work yet because the counters are empty.

Inside the for loops, there are if statements to check for anomalies:

* 1. In the first loop, if the count of requests from an IP exceeds the anomaly\_threshold, it prints a message indicating a high frequency of requests from that IP.
  2. In the second loop, if the status code is 500 or greater and its count exceeds the anomaly\_threshold, it prints a message indicating a high frequency of that status code in the responses. However, there's a syntax error in the print statement (} is misplaced).

**Script 3 :Generate automated email report(8)**

****



1. Import

| import smtplib  from email.mime.text import MIMEText  from email.mime.multipart import MIMEMultipart |
| --- |

* **import smtplib**: This line imports the SMTP (Simple Mail Transfer Protocol) library in Python. This library provides a way to send email messages from your Python script using an SMTP server.
* **from email.mime.text import MIMEText**: This line imports the MIMEText class from the email.mime.text module. MIMEText is used to create a text/plain MIME type message, which is a standard format for email messages.
* **from email.mime.multipart import MIMEMultipart**: This line imports the MIMEMultipart class from the email.mime.multipart module. MIMEMultipart is used to create a MIME multipart message, which allows you to include multiple parts (such as text, images, or attachments) in your email.(9)

1. function()

| # Function to send email  def send\_email(subject, message):  sender\_email = "your\_email@example.com"  receiver\_email = "recipient@example.com"  smtp\_server = "smtp.example.com"  smtp\_port = 587  smtp\_username = "your\_smtp\_username"  smtp\_password = "your\_smtp\_password"  msg = MIMEMultipart()  msg['From'] = sender\_email  msg['To'] = receiver\_email  msg['Subject'] = subject  msg.attach(MIMEText(message, 'plain'))  with smtplib.SMTP(smtp\_server, smtp\_port) as server:  server.starttls()  server.login(smtp\_username, smtp\_password)  server.send\_message(msg) |
| --- |

This code defines a function send\_email that takes a subject and a message as input and sends an email using SMTP. Let me break down the code for you:

* **sender\_email:** This variable holds the email address of the sender.
* **receiver\_email**: This variable holds the email address of the recipient.
* **smtp\_server:** This variable holds the address of the SMTP server.
* **smtp\_port:** This variable holds the port number of the SMTP server.
* **smtp\_username:** This variable holds the username for authenticating with the SMTP server.
* **smtp\_password:** This variable holds the password for authenticating with the SMTP server.

The function constructs an email message using MIMEMultipart and sets the sender, recipient, and subject fields. Then, it attaches the message body using MIMEText.

After that, the function establishes a connection to the SMTP server using smtplib.SMTP, starts TLS encryption with server.starttls(), and logs in with the provided username and password using server.login(). Finally, it sends the email message using server.send\_message().(9)

1. output

| # Generate email content  email\_subject = "Access Log Report"  email\_body = f"Number of occurrences of '404': {status\_count['404']}\n"  email\_body += f"Number of occurrences of '500': {status\_count['500']}\n"  # Display sorted IP addresses by occurrence  ip\_counts = Counter(ip\_addresses)  sorted\_ips = sorted(ip\_counts, key=ip\_counts.get, reverse=True)  email\_body += "Sorted IP addresses (most common to least):\n"  for ip in sorted\_ips:  email\_body += f"{ip} : {ip\_counts[ip]}\n"  # Optionally, include datetime details  email\_body += "Log entry dates and times:\n"  for dt in my\_datetimes:  email\_body += f"{dt.strftime('%Y-%m-%d %H:%M:%S')}\n"  # Send email  send\_email(email\_subject, email\_body) |
| --- |

***Generating Email Content****:*

* **email\_subject**: This variable holds the subject line for the email, which is set to "Access Log Report".
* **email\_body**: This variable holds the body of the email. It starts with the number of occurrences of HTTP status codes '404' and '500', followed by a sorted list of IP addresses based on their occurrence frequency, and optionally includes log entry dates and times.

***Displaying Sorted IP Addresses***:

* **ip\_counts**: This variable holds a Counter object that counts the occurrences of each IP address.
* **sorted\_ips**: This variable holds a list of IP addresses sorted based on their occurrence count, from most common to least common.
* The code iterates over the sorted IP addresses and appends each IP address along with its occurrence count to the email\_body.

***Optionally Including Datetime Details:***

If *my\_datetimes* is a list of datetime objects, the code iterates over them and appends their string representations (formatted as '%Y-%m-%d %H:%M:%S') to the email\_body.

***Sending Email:***

Finally, the send\_email function is called with the email\_subject and email\_body as arguments to send the email with the generated content.

**Expected Output**

For Output, we use the following grep, awk, and sort, uniq, cat , awl for parsing and extracting relevant information from the access logs.(10)

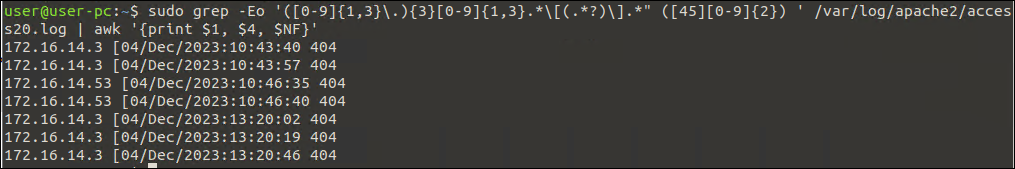
* **Sudo** : Stands for superdo. It allows users to do administrative tasks that require elevated permissions.
* **grep** : Stands for “**g**lobally search for a **r**egular **e**xpression and **p**rint matching lines.” It is used to search for lines in text files that match a specific pattern or keyword.
* **awk** : Stands for Aho, Winnie, and Knuth. Used to search, filter, and process text data based on patterns or conditions. Can perform calculations and manipulations on extracted data.
* **sort**: Sorts lines of text files in a specific order. By default, sorts lines lexicographically (alphabetically).
* **uniq** : Removes duplicate lines from a text file. It is very useful for cleaning up files that might contain redundant entries.
* **Cat** : A simple command used to display the contents of a file on the terminal. Can be used to concatenate (join) multiple files into one.
* **Chmod** : Stands for change mode. Used to modify the permissions associated with a file or directory.

**Commands in Use**

1. To extract lines matching the IP address, datetime, and status code pattern from the access log file.

**$sudo grep -Eo '([0-9]{1,3}.){3}[0-9]{1,3}.[(.?)].\*" ([45][0-9]{2}) ' /var/log/apache2/access20.log | awk '{print $1, $4, $NF}'**

* grep: This command is used to search for patterns in files or streams.
* -E: This option enables extended regular expressions.
* -o: This option tells grep to only output the matching part of the line, rather than the entire line.
* ([0-9]{1,3}\.){3}[0-9]{1,3}: This part of the pattern matches an IP address.
* [(.?)]: This matches a single character which is either (, . or ?.
* .\*: This matches any character (except for newline) zero or more times.
* " ([45][0-9]{2}) ": This part matches the HTTP status code. [45][0-9]{2} matches any number starting with 4 or 5 followed by two digits (e.g., 400-599).
* /var/log/apache2/access20.log: This is the path to the log file grep will search in.
* |: This is a pipe. It takes the output from the previous command and passes it as input to the next command.
* awk: This command is used for pattern scanning and processing.
* '{print $1, $4, $NF}': This is an awk command that tells it to print the first, fourth, and last fields of each line.
  + $1: First field (IP address)
  + $4: Fourth field (HTTP status code)
  + $NF: Last field (URL or resource accessed)

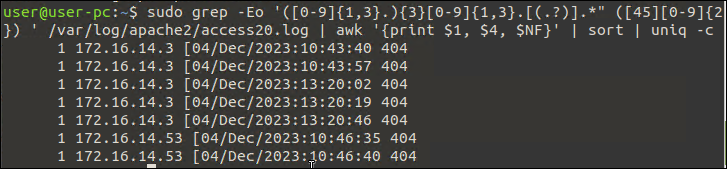


1. **$sudo grep -Eo '([0-9]{1,3}.){3}[0-9]{1,3}.[(.?)].\*" ([45][0-9]{2}) ' /var/log/apache2/access20.log | awk '{print $1, $4, $NF}' | sort | uniq -c**

This is almost similar to above with addition of sort and uniq -c.This command searches the Apache access log file for lines matching a certain pattern (IP address, HTTP status code, and URL/resource accessed), extracts those parts from each matching line, sorts the output, and then counts the occurrences of each unique line. This provides a summary of the log file data, showing how many times each combination of IP address, HTTP status code, and URL/resource accessed occurred.

* sort: This command sorts lines of text alphabetically or numerically. It's used here to sort the output of the awk command.
* uniq: This command filters adjacent matching lines from input and removes duplicate lines.
* -c: This option adds a count of the number of times each line occurred.

This combination counts the occurrences of each unique line in the sorted output.



1. **$ sudo grep -E '" [45][01235][0-9] [[:digit:]]{1,8} "' /var/log/apache2/access20.log**

This command will search for the logs with http status code which start with 4xx and 5xx.

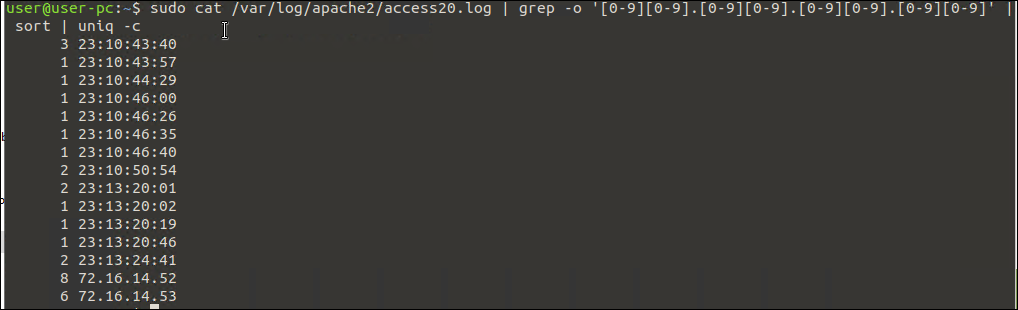
* ": Matches a double quote character.
* [45]: Matches either 4 or 5.
* [01235]: Matches any digit from 0 to 5.
* [0-9]: Matches any digit.
* [[:digit:]]: Matches any digit (equivalent to [0-9]).
* {1,8}: Matches between 1 and 8 occurrences of the preceding pattern.
* ": Matches a double quote character.



1. **$ sudo cat var/log/apache2/access20.log | grep -o '[0-9][0-9].[0-9][0-9].[0-9][0-9].[0-9][0-9]' | sort | uniq -c**

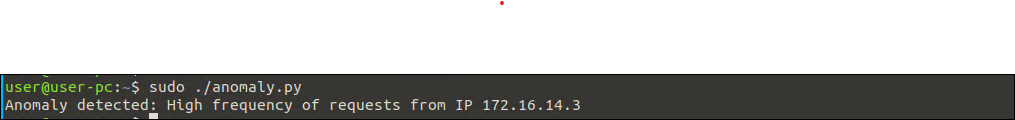
This command counts the occurrences of each unique IP address in the Apache access log file and its frequency.

* '[0-9][0-9].[0-9][0-9].[0-9][0-9].[0-9][0-9]': This pattern is looking for four groups of two digits separated by periods.



1. Anomaly.py - **$sudo ./anomaly.py**

This command is reading the final output from the file



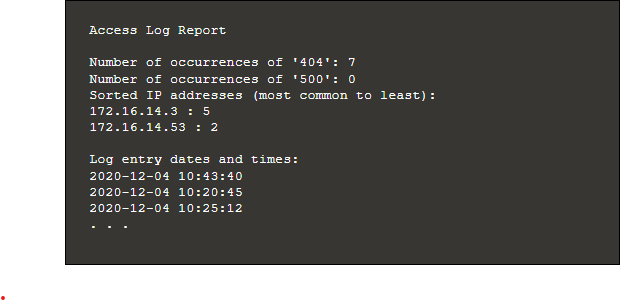
1. **$sudo awk ‘{print $1}’ /var/log/apache2/access20.log | sort | uniq -c |awl ‘$1 > 10’**

* sudo awk '{print $1}' /var/log/apache2/access20.log: This command extracts the first field from each line of the Apache access log file (/var/log/apache2/access20.log) using awk, which typically contains the IP address.
* | sort: The output of the awk command is piped (using |) to the sort command, which sorts the IP addresses in alphabetical order.
* | uniq -c: The sorted IP addresses are piped to uniq -c, which counts the occurrences of each unique IP address.
* | awk '$1 > 10': Finally, the output is piped to another awk command, which filters the output based on a condition ($1 > 10). This condition selects lines where the count (the first field) is greater than 10.



1. Email generation Result

**$cat ./email/\_generation.py**



**Documentation**

For every organization, it is very important to keep a documented version of the log Monitoring workflow. There could be multiple things that can be reported in documentation

1. **Log locations and access**

* The very first thing we need to know is where the logs are located for each server ( windows and linux)
* If there is any centralized logs, if so location of it
* Permissions, as logs, are a very crucial part of every organization especially when it comes to log monitoring for any kind of vulnerability. It is important that not all users have access to logs to avoid Unauthorized internal modifications.

1. **Workflow steps and explanation**

In our workflow we focused on 4 main things

* ***Data Collection*** : Monitor access logs daily from Both machines and/or web server. We used access .logs from Jumphost which are moved to our linux system. The location of logs in linux is /var/log/apache2/access20.log
* ***Data Processing*** : Implementation of script (such as Python or bash Script) to parse logs and set threshold. We used 2 key scripts here :
  + - * Data parsing script to extract relevant information from log files.
      * Script to calculate baseline traffic patterns and identify anomalies.
      * Script to generate automated email reports and send alerts.
* ***Alert Generation*** : We have set a Threshold = = 10 When the number of failed login attempts reaches the Threshold it will show the anomaly of IP address and will send an email to the manager if the number of failed attempts exceeds the predefined threshold.
* ***Weekly Email:*** Our last script is basically used for Compiling a email for manager, I have set it like the manager will get email of the generated report no matter if there is any unusual behavior recorded or not

3. **Script and commands**

The Script should be standard and should have comments added to make it readable for all users. The intention of doing this is to save time and make it more efficient so that next person does not have to create a script from scratch.

Similarly, commands should be simple as per company standard.

4. **Unusual behavior Report**

Although we have created a script, it's not the final thing. Based on the complexity and growth of the organization it is important to update script and document it if there are any changes in process.

**Unusual Behavior**

For New leaf organization we have following points which can represent unusual behavior(9)

* **Increased Failed Logins**: An unusual number of failed login attempts exceeding the predefined threshold can be an indication of Brute force attack
* **Suspicious IP Addresses**: Access attempts from suspicious or unfamiliar IP addresses could be an indication of Malicious redirect and DDoS attack.
* **Unusual Access Patterns**: Abnormal access patterns, such as login attempts outside of regular business hours could also indicate the Network intrusion attack which can lead to DoS
* **Unusual location Access**: Access from unusual geographic locations. For example outside Canada could also indicate a risk of potential Denial of service and data compromise.

**Potential Iterations**

No matter how good our script and techniques to get data from logs is, it’s never the best.there's always room for improvement in terms of generating responses that are more engaging, nuanced, and tailored to individual users' needs. Below are a few things that can be done to improve it.(11)

1. **Review your current workflow – and find bottleneck**

To determine how to improve your workflow, you should first get a sense of how your team manages its current processes. Identifying these challenges and roadblocks now means you can start creating solutions that address them. That will help to analyze which more could be done to make it more efficient.

1. **Real-Time Alerting**: Implement real-time alerting mechanisms to notify the analyst immediately upon detecting unusual behavior. Could also Implement anomaly detection algorithms to learn normal traffic patterns and automatically flag deviations.
2. **Enhanced Reporting:** Enhance the weekly report with visualizations and trend analysis to provide deeper insights into network traffic patterns and anomalies.
3. **Visualization Tools:** Utilize dashboards via tools such asSplunk and Kibana to visually represent login attempts and traffic patterns for easier analysis.
4. **Skill Development:**

* Educate yourself on data analysis techniques such as machine learning.
* Explore security information and event management (SIEM) solutions for comprehensive log management.
* Stay updated on evolving cyber threats and adapt the monitoring process accordingly.

**Conclusion**

By implementing this workflow, Turn a New Leaf can effectively monitor network traffic for unusual behavior, particularly focusing on detecting an unusual number of failed logins. The structured approach is outlined to help the New Leaf with finding any kind of vulnerabilities that could be associated with unusual login attempts so that they can take appropriate measures to tackle the risk.

**References**

1. Zeman, T. (2023, February 16). The importance of 24/7 cyber monitoring. Foresite. <https://foresite.com/blog/the-importance-of-24-7-cyber-monitoring/#:~:text=By%20monitoring%20and%20logging%20traffic,and%20respond%20to%20intrusions%20quickly>.
2. Welcome to Compass. (n.d.). <https://web.compass.lighthouselabs.ca/p/14/projects/log-monitoring>
3. Lyne, J. (2024, April 3). Security Best Practices for IT Project Managers | SANS Institute. <https://www.sans.org/white-papers/34257/>
4. Python, R. (n.d.). Python Tutorials – Real Python. <https://realpython.com/>
5. Python Tutorial. (n.d.). https://www.w3schools.com/python/default.asp
6. Anomaly detection with Python. (n.d.). Stack Overflow. <https://stackoverflow.com/questions/55639726/anomaly-detection-with-python>
7. Latest edition of Python Crash Course available this May. (n.d.). <https://nostarch.com/releases/PythonCrashCourse2ndEdition_PR.html>
8. Surameery, N. M. S., & Shakor, M. Y. (2023). Use Chat GPT to solve programming bugs. International Journal of Information Technology and Computer Engineering, 31, 17–22. <https://doi.org/10.55529/ijitc.31.17.22>
9. ChatGPT. (n.d.). <https://chat.openai.com/>
10. Weissig, J. (2014, July 14). CLI Monday: cat, grep, awk, sort, and uniq. sysadmincasts.com. <https://sysadmincasts.com/episodes/28-cli-monday-cat-grep-awk-sort-and-uniq>
11. <https://www.smartsheet.com/content/workflow-optimization#:~:text=Audit%20Your%20Workflow%3A%20Review%20each,working%20in%20a%20given%20process>.